DOCUMENT RESUME

ED 197 986 SE 034 161

AUTHOR Anderson, Lougenia; Gales, Larry
TITLE Programmer's Guide for FFORM. Physical Processes in Terrestrial and Aquatic Ecosystems, Computer Programs and Graphics Capabilities.
INSTITUTION Washington Univ., Seattle. Center for Quantitative Science in Forestry, Fisheries and Wildlife.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE May 78
GRANT NSF-GZ-2980; NSF-SED74-17696
NOTE 38p.; For related documents, see SE 034 160-167 and SE 033 581-597.

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Biological Sciences; *College Science; Computer Assisted Instruction; *Computer Graphics; *Computer Programs; Display Systems; Ecology; Higher Education; Interdisciplinary Approach; Mathematical Applications; Physics; *Programing; Science Education

ABSTRACT

This module is part of a series designed to be used by life science students for instruction in the application of physical theory to ecosystem operation. Most modules contain computer programs which are built around a particular application of a physical process. FFORM is a portable format-free input subroutine package written in ANSI Fortran IV which permits users to assign input values to program variables by name without regard to position or order. It is especially suitable for the input of directives which control the operation of interactive programs. FFORM is similar to the Namelist input/output systems found in many implementations of FORTRAN but features greatly superior error messages and error recovery procedures, and offers additional capabilities such as I/O, annotation of input, and echoing of input. A more detailed description of the purpose and use of FFORM is contained in its user's guide. (Author/CS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED197986

PHYSICAL PROCESSES IN TERRESTRIAL AND AQUATIC ECOSYSTEMS

COMPUTER PROGRAMS AND GRAPHICS CAPABILITIES

SCOPE OF INTEREST NOTICE
The ERIC Facility has assigned this document for processing to
In our judgement, this document is also of interest to the clearinghouses noted to the right. Index ing should reflect their special points of view.

PROGRAMMER'S GUIDE FOR SUBROUTINE FFORM

U.S. DEPARTMENT OF HEALTH EDUCATION & WELFARE NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY.

PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Patricia Babb
of the NSF
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

by

Lougenia Anderson and Larry Gales

CENTER FOR QUANTITATIVE SCIENCE IN
FORESTRY, FISHERIES AND WILDLIFE
University of Washington

SE 034 161

PROGRAMMER'S GUIDE FOR FFORM

by

Lougenia Anderson and Larry Gales

This instructional module is part of a series on Physical Processes in Terrestrial and Aquatic Ecosystems supported by National Science Foundation Training Grant No. GZ-2980

MAY 1978

JAN 1 9 1991

PROGRAMMER'S GUIDE FOR FFORM: A FORMAT-FREE INPUT SYSTEM

Identification

FFORM - A Format-Free Input Subroutine Package

Authors - Lougenia Anderson and Larry Gales

Date - May 1978, Center for Quantitative Science in Forestry, Fisheries and Wildlife, University of Washington, Seattle, Washington 98195

Purpose

FFORM is a portable format-free input subroutine package written in ANSI Fortran IV which permits users to assign input values to program variables by name without regard to position or order. For example, FFORM processes the following input card,

```
VELOC = 25.2, ARRAY(1) = 10 * 0 $
```

by assigning the value 25.2 to a program variable named VELOC and by zeroing out the first 10 elements of an array named ARRAY. FFORM is especially suitable for the input of directives which control the operation of interactive programs.

FFORM is similar to the Namelist input/output (henceforth input/output is abbreviated as I/O) systems found in many implementations of Fortran, but features greatly superior error messages and error recovery procedures, and offers additional capabilities such as text I/O, annotation of input, and echoing of input. Unlike Namelist, however, FFORM cannot interrogate tables within the Fortran compiler, and so requires auxiliary input information. This information consists of: 1) a restriction which limits input variables to a contiguous block of locations in blank common; 2) the starting address of the input block in blank common; and 3) a set of declarations which define the order, type, and

dimensionality of all input variables. The latter information, which duplicates the input variable declarations in the calling program, is passed to FFORM via a formatted file and provides the crucial link which enables it to construct symbol tables which control the processing of user input directives.

The input to FFORM consists of a set of blocks each of which is terminated by a dollar sign ($). Each block consists of one or more input expressions separated by commas and with blanks or comments freely interspersed. Each expression controls the input of values to a variable which may be of type integer, real, logical, text, double precision, or complex, and with zero, one, two or three dimensions. Repeated values are indicated by the syntax "n*v" where n is the repeat count and v is the value to be repeated, and comments are enclosed within slashes (/.../). The input is checked for, among other things, name and subscript errors which are automatically flagged and underscored. FFORM output is compatible with FFORM input in that the former can be read directly by the latter.

FFORM also permits a division of the input variable list into a number of named sublists, called io-lists, which limit I/O to variables in the sublist. Sublists need not be disjoint. The io-lists are especially useful for limiting FFORM output, but may be used for controlling input as well.

For a more detailed description of the purpose and use of FFORM, refer to its user's guide (Gales and Anderson 1978).

Usage

FFORM communicates with the calling program via argument lists, blank common, and files.

• Argument lists:

FFORM contains three main entry points named QQINTL, QQREAD, and QQWRIT whose arguments are unsubscripted integer variables or constants. QQINTL is used to initialize the system and to select io-lists, and must be called with a blank io-list before any other calls are made to FFORM. A call with a blank io-list name reads and processes the file which contains the declarations for the input variables and builds a symbol table. Subsequent calls to QQINTL with nonblank io-list names mark the symbol table to limit I/O to the variables specified in the io-list. Note that every call to QQINTL, with or without nonblank io-list names, re-reads the entire file which contains the declarations, hence the declaration file must remain intact. QQINTL is invoked by the following statement:

CALL QQINTL (WS, IOID, INFILE, ERFILE, ERROR)

where:

WS = the number of words occupied by the work space in blank common before the beginning of the input/output variable area (which is also in blank common);

IOID = a one-character identifier which matches one of the io-lists. Only those variables in the io-list so selected will be input when QQREAD is called or output when QQWRIT is called. If IOID is the blank character, then all variables specified in the declarations are available for input/output;

INFILE = a file containing the declarations and io-list(s);

ERFILE = a file on which error messages are written;

ERROR = an error flag which is set if an error is encountered; ERROR = 0 if no error was encountered.

Subroutine QQREAD reads in the user's input directives under the control of the io-list specified in the most recent call to QQINTL. If the io-list name is blank, it reads the next input block until it detects a dollar sign ($). If

the io-list name is nonblank, QQREAD first searches the input file for the characters "(X)" where "X" is the name of the sought for io-list. When found, it accepts input until it detects a dollar sign or the next io-list name. QQREAD is invoked by the following statement:

```
CALL QQREAD (INFILE, ECFILE, ERFILE, ERROR)
```

where:

INFILE = a file containing the user's free-form input directives;

ECFILE = a file on which input cards are echoed (printed out). If ECFILE is 0 the input is not echoed;

ERFILE = a file on which error messages are written;

ERROR = an error number which is set to a nonzero value if an error is detected; ERROR = 0 if no error was detected.

Subroutine QQWRIT prints out the names and values of input variables under control of the io-list specified in the most recent call to QQINTL. If the io-list name is blank, then QQWRIT outputs all variables; otherwise it outputs only those variables which belong to the current io-list. Output written by QQWRIT can be read by QQREAD. QQWRIT is invoked by the following statement:

```
CALL QQWRIT (OTFILE)
```

where:

OTFILE = a file on which output is to be written.

• Common blocks:

The free-form input system uses blank common and five labeled common blocks named /QQSCN/, /QQPARS/, /QQSYM/, /QQKHAR/, and /QQER/. All variables to be input or output by FFORM must occupy a contiguous block of locations in blank common. The displacement of the first input variable from the start of the blank common must be passed as an argument to subroutine QQINTL when FFORM is initialized. For example, suppose blank common is structured as follows:

```
COMMON // WSPACE(5400), A(10), V(2,5), WIND
```

where the variables A, V, and WIND are to be input via the free-form input system. In this case, when QQINTL is called, the WS argument must equal 5400 to indicate that the input/output variable block begins after the first 5400 words of blank common.* The five labeled common blocks /QQSCN/, /QQPARS/, /QQSYM/, /QQKHAR/, and /QQER/ are used only for internal operations in FFORM and can be ignored by the calling program.

• Files:

FFORM makes use of four files named INFILE, ERFILE, ECFILE, and OTFILE. For a call to subroutine QQINTL, INFILE is the unit number of a formatted file which contains the declarations and io-lists. It may contain any number of card images, each of which is up to 72 characters long. The format for INFILE when read by QQINTL is formally defined by the grammar in Appendix I and consists essentially of a set of data declarations which closely follow the declarations for variables in the calling program, except that: 1) continuation cards are not needed; 2) the declarations can begin at, or anywhere after, column 1; 3) integer variables in the calling program which are to receive alphanumeric text must be declared as TEXT, not INTEGER; and 4) the block of declarations is terminated by "$ io-lists $." If no io-lists are present then the declaration block is terminated by $$. Each io-list starts on a new card and is of the form (X) v, v,...,v where X is a one-character io-list name and v, v,...,v are the names of input variables. It is <u>absolutely crucial</u> that the declarations on INFILE preserve the exact <u>order</u>, <u>type</u>, and <u>dimensionality</u> of the input variables as they occur in blank common in the calling program. However, variables mentioned in the io-lists may appear in any order.

*However, on many computer systems a single real number occupies n computer words where n is typically 2 or 4. In this case WS must equal n*5400.

The format for INFILE is illustrated in the following example. Suppose that CATCH, TITLE, ENERGY, and LIVE are input variables declared in the main program as follows:

```
COMMON // WSPACE(5400), CATCH(10,5), TITLE(60)
*          ENERGY, LIVE
INTEGER CATCH, TITLE
REAL ENERGY
LOGICAL LIVE
```

Suppose that TITLE is to contain alphanumeric text, one character per word, and the the following io-lists are provided: (A) TITLE, LIVE (B) ENERGY, CATCH and (C) CATCH, LIVE; then INFILE must be formatted as follows:

```
INTEGER CATCH(10,5)
TEXT TITLE(60)
REAL ENERGY
LOGICAL LIVE $
(A) TITLE, LIVE
(B) ENERGY, CATCH
(C) CATCH, LIVE $
```

INFILE is automatically rewound by QQINTL at the start of execution.

For a call to QQREAD, INFILE is the unit number (it may be a different number than in the call to QQINTL) of the file containing the free-form input. Again, it may contain any number of card images, each of which is up to 72 characters long. The format of INFILE when read by QQREAD is formally defined in Appendix II and essentially consists of sets of [name] = [value] expressions, where [name] is a simple or subscripted variable and [value] is a value or a list of (possibly repeated) values. For example, the user input for the variables in the above example might consist of several input blocks on INFILE formatted as follows:

```
LIVE = T, CATCH(1,5) = 10*3 $
TITLE = # CATCH OF LOBSTER #, CATCH(1,6) = 7, 8, 14, 2*0,
ENERGY = 6.32E+02 $
(A) LIVE = F $
(C) LIVE = T $
```

INFILE is not rewound by QQREAD. Note that text variables are delimited by the # sign, and that each input sublist must begin on a new card.

ERFILE, ECFILE, and OTFILE are the unit numbers of formatted output files written by the free-form input system. ERFILE contains any error messages generated during execution; ECFILE, if nonzero, contains an echo of each input card image read by QQREAD; OTFILE contains the card images of output for variables generated by a call to QQWRIT. The output file OTFILE is in a form compatible with free-form input specifications and can thus be reread by a call to QQREAD (the file must first be rewound, however).

The characteristics of the files used by the free-form system are summarized as follows:

FILE NAME	AUTOMATIC REWIND	READ BY QQINTL	READ BY QQREAD	WRITTEN BY FFORM SYSTEM	UNIQUE UNIT NO.
INFILE	(By QQINTL only)	Yes	Yes	No	Yes
ECFILE	No	No	No	Yes	No
ERFILE	No	No	No	Yes	No
OTFILE	No	No	No	Yes	No

The column labeled "AUTOMATIC REWIND" means that the file is rewound at the beginning of execution. The column labeled "UNIQUE UNIT NO." specifies whether or not different file names may reference the same unit number. Only INFILE need be unique in the free-form system. Thus, it may be the case that ECFILE = ERFILE = OTFILE.

The free-form input system does not check the files to see if file names reference valid unit numbers. This type of error will generally trigger error messages and actions which are peculiar to a given computer installation.

Structure

The structure for FFORM is shown by the parse tree in Appendix I.

Subroutines

The following is a list and brief description of all subroutines in FFORM in alphabetical order:

QQBLOK: Assigns constant values to all variables which appear in labeled common; it is here that the word size of all variable types must be set or changed for different implementations of the free-form system.

QQCVAL: Called by QQVEXP to parse a single [complex-value] from the input and to assign the value parsed [repeat-value] number of times to the current (complex) variable.

QQCWRT: Called by QQWRIT to output all the values associated with the current (complex) variable; output is in a form compatible with free-form input specifications.

QQDECL: Called by QQDELS to parse a single [declaration]; determines the type which is indicated by a reserved word and calls QQVLST to parse the [variable-list].

QQDELS: Called by QQINTL to parse the [declaration-list]; it calls QQDECL repeatedly to parse all the [declaration]s until an [end-of-file] is encountered.

QQDGT: Returns the real value associated with a given numeric character.

QQDVAL: Called by QQVEXP to parse a single [double-precision-value] from the input and to assign it [repeat-value] number of times to the current (double precision) variable.

QQDWRT: Called by QQWRIT to output all the values associated with the current (double precision) variable; output is in a form compatible with free-form input specifications.

QQERR: Outputs an error message containing a given error number and reads from the input file until an [end-of-file] is encountered.

QQFIND: Returns a pointer to an item in the symbol table if the global NAME array matches one of the variable names in the symbol table; zero otherwise.

QQFIOL: Scans the input until the selected [io-name] is found.

QQGNAM: Moves the variable name just scanned into the global NAME array.

QQIEXP: Called by QQINLS to parse a single [input-exp]; it calls QQNPAR to parse the [name-part], parses the equal sign, and then calls QQVPAR to parse the [value-part].

QQINLS: Called by QQREAD to parse the [input-list]. It repeatedly calls QQIEXP to parse each [input-exp] until an [end-of-file] is encountered.

QQINST: Makes an entry in the symbol table for the variable which has just been parsed in the declarations.

QQINTL: Initializes the free-form input package, calls QQDELS to parse the declarations, and calls QQIOLS to parse the [io-list]s.

QQIOLS: Called by QQINTL to parse the [io-list] selected; it calls QQFIOL to find the [io-name] specified and then calls QQVFLG to parse and flag those variables which follow.

QQIVAL: Called by QQVEXP to parse a single [integer-value] from the input and to assign it [repeat-value] number of times to the current (integer) variable.

QQIWRT: Called by QQWRIT to output all the values associated with the current (integer) variable; output is in a form compatible with free-form input specifications.

QQKLS: Returns the partition class (alpha, numeric, special character, or other) associated with the k^{th} character in the KHAR array. This function uses an algorithm for character comparison that is machine-dependent. The ordering of the internal representation of characters is used to establish the partition and must be changed for different orderings of characters.

QQLEX: Lexical analyzer; acts on the KHAR array which contains the current card image to determine the next syntactic symbol.

QQLVAL: Called by QQVEXP to parse a single [logical-value] from the input and to assign it [repeat-value] number of times to the current (logical) variable.

QQLWRT: Called by QQWRIT to output all the values associated with the current (logical) variable; output is in a form compatible with free-form input specifications.

QQNPAR: Called by QQIEXP to parse the [name-part] of a free-form [input-expression].

QQNUM: Converts numeric string contained in KHAR array into its numeric value, either integer, real, or double precision.

QQREAD: Parses free-form input, setting variables located in blank common to the values specified; it is assumed that the symbol table has been filled with the necessary information by a prior call to QQINTL.

QQRVAL: Called by QQVEXP to parse a single [real-value] from the input and to assign it [repeat-value] number of times to the current (real) variable.

QQRWRT: Called by QQWRIT to output all the values associated with the current (real) variable; output is in a form compatible with free-form input specifications.

QQSCAN: Free-form scanner; drives the lexical analyzer QQLEX and maintains the necessary information for the current and look-ahead syntactic symbols.

QQSLST: Called by QQVAR and by QQNPAR to parse a [subscript-list].

QQTVAL: Called by QQVEXP to parse a single [text-value] from the input and to assign it [repeat-value] number of times to the current (integer) variable.

QQTWRT: Called by QQWRIT to output all the values associated with the current (text) variable; output is in a form compatible with free-form input specifications.

QQVAR: Called by QQVLST to parse the [variable-part] of declarations.

QQVEXP: Called by QQVPAR to parse a single [value-exp]; it checks for a [repeat-value] and calls the appropriate subroutine to parse the value itself.

QQVFLG: Called by QQIOLS to parse the [var-part] of the selected [io-list] and to set the symbol table flag associated with each variable so named.

QQVLST: Called by QQDECL to parse a [variable-list].

QQVPAR: Called by QQIEXP to parse the [value-part] of an [input-exp].

QQVRW: Determines if a character string in KHAR array is a variable name or a reserved word.

QQWRIT: Driving routine to output all variables which appear in the current [io-list] in a form compatible with free-form input.

Coding information

• Literals and constants:

The literals used in the free-form system can be divided into the following classes:

1) The integers 1-25 are used as error numbers;
2) the integers 0 and 1 are used as flags in the symbol table;
3) the integers 1 and 2 are used to indicate a reference to the current scanner token and look-ahead scanner token, respectively;
4) the integer 0 is used as a flag to indicate no error conditions exists;
5) the integers 1-72 are used as subscripts in referencing various arrays;
6) the integers 0 and 1, the real number 0., and the double precision number 0.D0 are used in initialization;
7) the integer 1 is used as an increment/decrement;
8) the logical constants .TRUE. and .FALSE.;
9) the integer -1, the real number 10. and the double precision numbers 1.D1 and -1.D0 are used in converting a string value to a numeric value.

Constants are divided into two classes:

1) those values that are assigned in the"CONSTANTS" section of each routine;
2) those values which, while assigned in the "INITIALIZATION" section of each routine, usually remain constant throughout execution of the entire free-form system (they can be changed during execution but are more constant than most variables).

The two classes of constants are described in the following tables:

CONSTANTS (Class 1)

NAME	VALUE	SUB.	BLOCK	DESCRIPTION
RSWL(*,*)	(see description)	QQVRW	--	Reserved word table; contains all reserved words recognized by FFORM system.
NRSW	7	QQVRW	--	Number of reserved words contained in RSWL.
LRSW	9	QQVRW	--	Length of longest reserved word in RSWL.
NUM(*)	0,1,2,3,4,5, 6,7,8,9	QQDGT	--	Number table; contains all digits as characters.
ISIZE	1	QQBLOK	/QQSYM/	Number of words occupied by an integer value.
RSIZE	1	QQBLOK	/QQSYM/	Number of words occupied by a real value.
LSIZE	1	QQBLOK	/QQSYM/	Number of words occupied by a logical value.
CSIZE	2	QQBLOK	/QQSYM/	Number of words occupied by a complex value.
DSIZE	2	QQBLOK	/QQSYM/	Number of words occupied by a double precision value.
NLIM	6	QQBLOK	/QQSYM/	Maximum length of a variable name, in characters.
NAME(*)	1HI,1HR,1HL, 1HT,1HC,1HD, 1HD	QQVRW	--	Contains first letter of reserved words in RSWL array.
NVAR	4	QQIWRT	--	The number of values to be written on one line of output.
	20	QQLWRT	--	
	4	QQRWRT	--	
	2	QQCWRT	--	
	4	QQDWRT	--	
	56	QQTWRT	--	
AA,BB...ZZ	1HA,1HB...1HZ	QQBLOK	/QQKHAR/	Contains all the letters in the alphabet (A-Z).

NAME	VALUE	SUB.	BLOCK	DESCRIPTION
PLUS	1H+	QQBLOK	/QQKHAR/	Plus sign character.
MINUS	1H-	QQBLOK	/QQKHAR/	Minus sign character.
STAR	1H*	QQBLOK	/QQKHAR/	Star Character.
SLASH	1H/	QQBLOK	/QQKHAR/	Slash character.
LPAREN	1H(	QQBLOK	/QQKHAR/	Left parenthesis character.
RPAREN	1H)	QQBLOK	/QQKHAR/	Right parenthesis character.
EQUAL	1H=	QQBLOK	/QQKHAR/	Equal sign character.
BLANK	1H	QQBLOK	/QQKHAR/	Blank character.
COMMA	1H,	QQBLOK	/QQKHAR/	Comma character.
PERIOD	1H.	QQBLOK	/QQKHAR/	Period character.
POUND	1H#	QQBLOK	/QQKHAR/	Pound sign character.
EDFMK	1H$	QQBLOK	/QQKHAR/	Dollar sign character/end-of-file mark.
ALPHA	1HA	QQBLOK	/QQKHAR/	Signifies alphabetic character partition class.
NUMRL	1H0	QQBLOK	/QQKHAR/	Signifies numeric character partition class.
SPCHR	1H+	QQBLOK	/QQKHAR/	Signifies special character partition class.
OTHER	1H'	QQBLOK	/QQKHAR/	Signifies unrecognizable character partition class.
		CONSTANTS (Class 2)		
		(Relatively constant constants)		
SLIM	50	QQINTL	/QQSYM/	Maximum number of entries which can be made in symbol table (at end of execution of QQINTL, SLIM is set to the number of entries which were made in the symbol table).
KSTART	0	QQINTL QQREAD	/QQPARS/	The column number minus 1 at which scanning will commence when a new card image is read by the scanner.

NAME	VALUE	SUB.	BLOCK	DESCRIPTION
KEND	72	QQINTL QQREAD	/QQPARS/	The last column on the card image that will be scanned by the scanner before a new card image is read.

• Word lengths:

Six types of variables can be input or output by the free-form system: INTEGER, REAL, LOGICAL, TEXT, DOUBLE PRECISION, and COMPLEX. For a particular implementation, the number of words occupied by a single value of each type must be specified by setting the constants ISIZE, RSIZE, LSIZE, DSIZE, and CSIZE to the number of words occupied by a single value of the corresponding type (it is assumed that a single text character occupies the same amount of storage as a single integer value--hence there is no need for a TSIZE constant). The constants ISIZE...CSIZE are all of type integer themselves as the free-form system will only handle values that occupy integer multiples of whole words of storage. Thus it is possible for a double precision value to occupy four words of storage (DSIZE must be set to four in this case) but it is not possible to input a value for a variable that is declared to be a half-word integer variable. This restriction is necessary so that, when storing input values or fetching values to be output, blank common is addressed in a uniform manner which maintains ANSI standards and the transportability of the free-form package.

• Naming conventions:

All subroutine, function, and common block names within FFORM start with the letters QQ in order to minimize conflict with user or system routines. All common block locations which are unused in a given subroutine are represented by dummy variables of the form ZZZn or ZZnn where n is a digit. Each dummy variable may span the area occupied by a number of variables or arrays.

. Machine dependencies occur in subroutines QQBLOK, QQKLS, QQINTL, QQVEXP, and QQWRIT as follows:

QQBLOK: This routine initializes the constants ISIZE, RSIZE, LSIZE, CSIZE, and DSIZE, which determine the number of words of storage occupied by a value of the corresponding type and must be set to match a particular implementation.

QQKLS: This function uses a machine-dependent algorithm for character comparison.

QQINTL: QQVEXP QQWRIT These routine redefine blank common, which is allocated in the calling (user) program, to be only one word long. This will cause a problem in computer systems which require all occurrences of common to be of the same length.

Limitations

FFORM is subject to the following limitations: 1) all variable names must start with a letter and are limited to six alphanumeric characters; 2) the maximum number of input variables which FFORM can handle at any one time is 50; 3) all text information is stored one character per word; and 4) no check is made to see if file unit numbers are valid--such errors are left to the computer system.

Error Handling

FFORM checks for 25 error conditions. If any of these conditions occurs, it outputs an appropriate error number, underscores that part of the input card in error, skips down the input file to the end of an input block, indicated by a dollar sign ($), and returns to the calling program with ERROR set to the appropriate nonzero value. The error messages are listed in Appendix III.

Extensions

Symbol table: the symbol table is currently accessed by a simple linear search. If its size is substantially increased, it might be worthwhile to use a hashing algorithm instead. The only changes would involve subroutines QQINST (symbol table insertion) and QQFIND (symbol table lookup).

Octal values: the current implementation does not permit octal values as input. It is only necessary to change subroutine QQNUM in order to read octal values.

Scanner extraction: the free-form scanner can be extracted and used in a different environment. It includes the routines QQSCAN, QQLEX, QQERR, QQKLS, QQNUM, and QQVRW.

Computer resources

• Storage:

The object deck for the free-form input system occupies 7124 (octal) words of storage when compiled under the CDC 6400 Fortran Extended compiler, optimization level 2.

• Execution time:

The execution time for the free-form system depends primarily on the number of tokens which must be parsed to interpret the declarations and the input directives (examples of tokens: a reserved word, a variable name, a left parenthesis, a numeric value, a text constant, a comma, etc.). It takes, on the average, approximately 2×10^{-3} CPU seconds per token on the CDC 6400 computer. For example, a short sample run using the free-form system contained about 100 "free-form input tokens" and required 0.195 CPU seconds to execute. Hence, if the average input card contains about 15 tokens, then FFORM will process about 30 cards per second.

Sample runs

The annotated listings on the next few pages illustrate the control, program, and input data cards, plus the associated output generated by the programs, for two sample runs.

```
XFF1S,T10,CM45000.                          SMALL FREE FORM INPUT EXAMPLE
ACCOUNT,XXXXX XX,XXXXXX.
MNF,J.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE MNF CARD CALLS THE FORTRAN COM-  *
COMMENT.* PILER TO COMPILE THE SAMPLE EXECU-   *
COMMENT.* TION PROGRAM.                        *
COMMENT.*****************************************
COMMENT.
COPYBR,INPUT,DFILE.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE COPYBR CARD COPIES THE DECLARA-  *
COMMENT.* TION FILE, WHICH FOLLOWS THE MAIN    *
COMMENT.* PROGRAM, TO THE LOGICAL FILE NAMED   *
COMMENT.* DFILE. THIS IS NECESSARY AS THE      *
COMMENT.* SUBROUTINE QQINTL AUTOMATICALLY      *
COMMENT.* REWINDS THE DECLARATION FILE.        *
COMMENT.*****************************************
COMMENT.
ATTACH,BFF,ID=BFF.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE ABOVE CARD ATTACHES THE FREE     *
COMMENT.* FORM INPUT SYSTEM.                    *
COMMENT.*****************************************
COMMENT.
LOAD,LGO.
LOAD,BFF.
EXECUTE,FFIEX,INPUT,OUTPUT,DFILE.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE THREE ABOVE CARDS LOAD AND EXE-  *
COMMENT.* CUTE THE SAMPLE PROGRAM.              *
COMMENT.*****************************************
COMMENT.
*EOR
      PROGRAM FFIEX(INPUT,OUTPUT,TAPE1,TAPE5=INPUT,TAPE6=OUTPUT)
C
C SIMPLE EXAMPLE OF THE USE OF THE FREE FORM INPUT SYSTEM
C
      COMMON // K(2,2,2), A(20), B, BFLAG(3), X, Y, Z,
     .          AREA, LEN, WIDTH
      INTEGER   K, A
      REAL      B
      LOGICAL   BFLAG
      COMPLEX   X, Y, Z
      DOUBLE PRECISION  AREA, LEN, WIDTH
C
C
      INTEGER   WS, IOID, ERROR
      INTEGER   DCF, INF, ERF, OTF, ECF
C
C
      DATA       WS,      IOID,    ERROR
     .         / 0,       1H ,       0 /
      DATA       DCF,     INF,     ERF,     OTF,     ECF
     .         / 1,         5,       6,       6,       6 /
C
```

```
C
      CALL QQINTL(WS,IOID,DCF,ERF,  ERROR)
      WRITE(6,1)
    1 FORMAT(42H1---FREE FORM INPUT AS ECHOED BY QQREAD--- // 1HO)
      IF(ERROR .EQ. 0)CALL QQREAD(INF,ECF,ERF,  ERROR)
      WRITE(6,2)
    2 FORMAT(1HO,/,34HO---OUTPUT AS WRITTEN BY QQWRIT--- // 1HO)
      IF(ERROR .EQ. 0)CALL QQWRIT(OTF  )
      STOP
      END
*EOR
INTEGER K(2,2,2)
TEXT A(20)
REAL B
LOGICAL BFLAG(3)
COMPLEX X,Y,Z
DOUBLE PRECISION AREA,
LEN, WIDTH $
$
*EOR
   /     DATE:  JUNE 9, 1977     /
   A = ±AREA IN SQUARE MILES±,   B = 40E2,
   K(1,1,1) = 4*3, 60, -9,
              755,
   BFLAG(3) = .T.,    X = (4.,.975),
   BFLAG(2) = F,      Y = (7E4,3.9),
   BFLAG(1) = .TRUE., Z = (99.65,1.),
   K(2,2,2) = 4522,
   LEN = 65E6,        WIDTH = 44.3,
   AREA = .1D-6     $
*EOR
*EOF
```

---FREE FORM INPUT AS ECHOED BY QQREAD---

```
/     DATE:  JUNE 9, 1977     /
A = *AREA IN SQUARE MILES*,   B = 40E2,
K(1,1,1) = 4*3, 60, -9,
           755,
BFLAG(3) = .T.,    X = (4.,.975),
BFLAG(2) = F,      Y = (7E4,3.9),
BFLAG(1) = .TRUE., Z = (99.65,1.),
K(2,2,2) = 4522,
LEN = 65E6,        WIDTH = 44.3,
AREA = .1D-6      $
```

---OUTPUT AS WRITTEN BY QQWRIT---

```
K      =              3,              3,              3,              3,
                     60,             -9,            755,           4522,
A      =  *AREA IN SQUARE MILES*,
B      =   .4000000E+04,
BFLAG  =  T, F, T,
X      =  ( .4000000E+01, .9750000E+00),
Y      =  ( .7000000E+05, .3900000E+01),
Z      =  ( .9965000E+02, .1000000E+01),
AREA   =   .1000000D-06,
LEN    =   .6500000D+08,
WIDTH  =   .4430000D+02,
$
```

```
XFFIL,T10,CM47000.                          LARGE FREE FORM INPUT EXAMPLE
ACCOUNT,XXXXXXXX,XXXXXX.
MNF,J.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE MNF CARD CALLS THE FORTRAN COM-  *
COMMENT.* PILER TO COMPILE THE SAMPLE EXECU-   *
COMMENT.* TION PROGRAM.                        *
COMMENT.*****************************************
COMMENT.
COPYBR,INPUT,DFILE.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE COPYBR CARD COPIES THE DECLARA-  *
COMMENT.* TION FILE, WHICH FOLLOWS THE MAIN     *
COMMENT.* PROGRAM, TO THE LOGICAL FILE NAMED   *
COMMENT.* DFILE. THIS IS NECESSARY AS THE      *
COMMENT.* SUBROUTINE QQINTL AUTOMATICALLY      *
COMMENT.* REWINDS THE DECLARATION FILE.        *
COMMENT.*****************************************
COMMENT.
ATTACH,BFF,ID=BFF.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE ABOVE CARD ATTACHES THE FREE     *
COMMENT.* FORM INPUT SYSTEM.                    *
COMMENT.*****************************************
COMMENT.
LOAD,LGO.
LOAD,BFF.
EXECUTE,FFIEX,INPUT,OUTPUT,DFILE.
COMMENT.
COMMENT.*****************************************
COMMENT.* THE THREE ABOVE CARDS LOAD AND EXE-  *
COMMENT.* CUTE THE SAMPLE PROGRAM.              *
COMMENT.*****************************************
COMMENT.
*EOR
      PROGRAM FFIEX(INPUT,OUTPUT,TAPE1,TAPE5=INPUT,TAPE6=OUTPUT)
C
C LONGER EXAMPLE OF THE USE OF THE FREE FORM INPUT SYSTEM
C
      COMMON // K(2,2,2), A(20), B, BFLAG(3), X, Y, Z,
     .          AREA, LEN, WIDTH, COM(50,4), SAVE(4,5,6),
     .          DIST(3,3), TIME(3,3), VEL(3,3), DX, DY, DZ,
     .          XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX,
     .          XRATE, YRATE, ZRATE, MEAN(5), SD(5), AVG(5)
      INTEGER   COM
      REAL      SAVE
      DOUBLE PRECISION DIST, TIME, VEL
      COMPLEX   DX, DY, DZ
      INTEGER   XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX
      REAL XRATE, YRATE, ZRATE
      DOUBLE PRECISION MEAN, SD, AVG
      INTEGER   K, A
      REAL      B
      LOGICAL   BFLAG
      COMPLEX   X, Y, Z
      DOUBLE PRECISION  AREA, LEN, WIDTH
C
```

```
C
      INTEGER        WS,      IOID,    IOIDA,   IOIDB,   ERROR
      INTEGER    DCF, INF, ERF, OTF, ECF
C
C
      DATA           WS,      IOID,    IOIDA,   IOIDB,   ERROR    /
     .               0,       1H ,     1HA,     1HB,       0      /
      DATA       DCF,     INF,     ERF,     OTF,     ECF
     .          / 1,       5,       6,       6,       6 /
C
C
      CALL QQINTL(WS,IOID,DCF,ERF,  ERROR)
      WRITE(OTF,1)
    1 FORMAT(42H1---FREE FORM INPUT AS ECHOED BY QQREAD--- // 1HO)
      CALL QQREAD(INF,ECF,ERF,  ERROR)
      WRITE(OTF,2)
    2 FORMAT(1HO,/,34HO---OUTPUT AS WRITTEN BY QQWRIT--- // 1HO)
      CALL QQWRIT(OTF  )
      CALL QQINTL(WS,IOIDA,DCF,ERF,  ERROR)
      WRITE(OTF,12) IOIDA
   12 FORMAT(23HO---OUTPUT FOR SUBLIST ,1A1,
     .        24H AS WRITTEN BY QQWRIT--- // 1HO)
      CALL QQWRIT(OTF  )
      CALL QQINTL(WS,IOIDB,DCF,ERF,  ERROR)
      WRITE(OTF,11)
   11 FORMAT(53HO---FREE FORM INPUT FOR SUBLIST B AS ECHOED BY QQREAD
     .       3H--- // 1HO)
      CALL QQREAD(INF,ECF,ERF,  ERROR)
      WRITE(OTF,12) IOIDB
      CALL QQWRIT(OTF  )
      STOP
      END
*EOR
INTEGER K(2,2,2)
TEXT A(20)
REAL B
LOGICAL BFLAG(3)
COMPLEX X,Y,Z
DOUBLE PRECISION AREA,
LEN, WIDTH
TEXT COM(50,4)
REAL SAVE(4,5,6)
DOUBLE PRECISION DIST(3,3), TIME(3,3), VEL(3,3)
COMPLEX DX, DY, DZ
INTEGER XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX
REAL XRATE, YRATE, ZRATE
DOUBLE PRECISION MEAN(5), SD(5), AVG(5)
$
(A) ZMIN, ZMAX, YRATE, BFLAG
(B) DX, DY, AVG
$
*EOR
   /     DATE:  JUNE 9, 1977     /
   AVG = 4*3.E+2, MEAN = 4*65.6666,  SD = 4*.0015,
   XRATE = 65.1, YRATE = 1, ZRATE = 99.676,
   MEAN(5) = 4D4, SD(5) = 5E5, AVG(5) = 26.001D-10,
   COM(1,1) = 'X AXIS-TIME IN MICROSECONDS                          ',
   COM(1,2) = 'Y AXIS-VELOCITY AS A FUNCTION OF TIME                ',
   COM(1,3) = 'DATE AND TIME:  JUNE 13, 1977   9:52 A.M.            ',
   COM(1,4) = 'Y AXIS-DISTANCE AS A FUNCTION OF TIME                ',
```

```
   XMAX = 500, YMAX = 24,  ZMAX = 27,
   XMIN = 0, YMIN = 1,   ZMIN = -3,
   DX = (40.,-36.), DY = (1.376, 99.000001), DZ = (-676.1,93.2),
   DIST(1,1) = 9*0., TIME = 9*0., VEL = 6*0.,
   3*1.,
   SAVE = 40*999.999,
   40 * -111.111, 40   *    999.999,
   A = =AREA IN SQUARE MILES=,   B = 40E120,
   K(1,1,1) = 4*3, 60, -9,
              755,
   BFLAG(3) = .T.,   X = (4.,.975),
   BFLAG(2) = F,     Y = (7E4,3.9),
   BFLAG(1) = .TRUE., Z = (99.65,1.),
   K(2,2,2) = 4522,
   LEN = 65E6,       WIDTH = 44.3,
   AREA = .1D-6     $
   (B) DX = (20.,-18.), DY = (2.752,198.000002),
       AVG = 5*18.E+10,     $
*EOR
*EOF
```

---FREE FORM INPUT AS ECHOED BY QQREAD---

```
/     DATE:  JUNE 9, 1977     /
AVG = 4*3.E+2, MEAN = 4*65.6666,  SD = 4*.0015,
XRATE = 65.1, YRATE = 1, ZRATE = 99.676,
MEAN(5) = 4D4, SD(5) = 5E5, AVG(5) = 26.001D-10,
COM(1,1) = =X AXIS-TIME IN MICROSECONDS                              =,
COM(1,2) = =Y AXIS-VELOCITY AS A FUNCTION OF TIME                    =,
COM(1,3) = =DATE AND TIME:  JUNE 13, 1977   9:52 A.M.                =,
COM(1,4) = =Y AXIS-DISTANCE AS A FUNCTION OF TIME                    =,
XMAX = 500, YMAX = 24,  ZMAX = 27,
XMIN = 0, YMIN = 1,   ZMIN = -3,
DX = (40.,-36.), DY = (1.376, 99.000001), DZ = (-676.1,93.2),
DIST(1,1) = 9*0., TIME = 9*0., VEL = 6*0.,
3*1.,
SAVE = 40*999.999,
40 * -111.111, 40   *    999.999,
A = =AREA IN SQUARE MILES=,   B = 40E120,
K(1,1,1) = 4*3, 60, -9,
           755,
BFLAG(3) = .T.,    X = (4.,.975),
BFLAG(2) = F,      Y = (7E4,3.9),
BFLAG(1) = .TRUE., Z = (99.65,1.),
K(2,2,2) = 4522,
LEN = 65E6,        WIDTH = 44.3,
AREA = .1D-6     $
```

---OUTPUT AS WRITTEN BY QQWRIT---

```
K      =              3,             3,             3,             3,
                     60,            -9,           755,          4522,
A      =  =AREA IN SQUARE MILES=,
B      =   .4000000+122,
BFLAG  =  T, F, T,
X      =  (  .4000000E+01,  .9750000E+00),
Y      =  (  .7000000E+05,  .3900000E+01),
Z      =  (  .9965000E+02,  .1000000E+01),
AREA   =   .1000000D-06,
LEN    =   .6500000D+08,
WIDTH  =   .4430000D+02,
COM    =  =X AXIS-TIME IN MICROSECONDS                        Y AXIS=,
          =-VELOCITY AS A FUNCTION OF TIME              DATE AND TIM=,
          =E:  JUNE 13, 1977   9:52 A.M.          Y AXIS-DISTANCE AS=,
          = A FUNCTION OF TIME              =,
SAVE   =   .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
```

```
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
          -.1111110E+03, -.1111110E+03, -.1111110E+03, -.1111110E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
           .9999990E+03,  .9999990E+03,  .9999990E+03,  .9999990E+03,
DIST    =  0.          , 0.          , 0.          , 0.          ,
           0.          , 0.          , 0.          , 0.          ,
           0.          ,
TIME    =  0.          , 0.          , 0.          , 0.          ,
           0.          , 0.          , 0.          , 0.          ,
           0.          ,
VEL     =  0.          , 0.          , 0.          , 0.          ,
           0.          , 0.          ,  .1000000D+01,  .1000000D+01,
           .10000000+01,
DX      = ( .4000000E+02,-.3600000E+02),
DY      = ( .1376000E+01, .9900000E+02),
DZ      = (-.6761000E+03, .9320000E+02),
XMIN    =             0,
XMAX    =           500,
YMIN    =             1,
YMAX    =            24,
ZMIN    =            -3,
ZMAX    =            27,
XRATE   =   .6510000E+02,
YRATE   =   .1000000E+01,
ZRATE   =   .9967600E+02,
MEAN    =   .6566660D+02,  .6566660D+02,  .6566660D+02,  .6566660D+02,
           .4000000D+05,
SD      =   .1500000D-02,  .1500000D-02,  .1500000D-02,  .1500000D-02,
           .5000000D+06,
AVG     =   .3000000D+03,  .3000000D+03,  .3000000D+03,  .3000000D+03,
           .2600100D-08,
$

---OUTPUT FOR SUBLIST A AS WRITTEN BY QQWRIT---


BFLAG   =  T, F, T,
ZMIN    =           -3,
ZMAX    =           27,
YRATE   =   .1000000E+01,
$

---FREE FORM INPUT FOR SUBLIST B AS ECHOED BY QQREAD---
```

```
(B) DX = (20.,-18.), DY = (2.752,198.000002),
    AVG = 5*18.E+10,      $
```

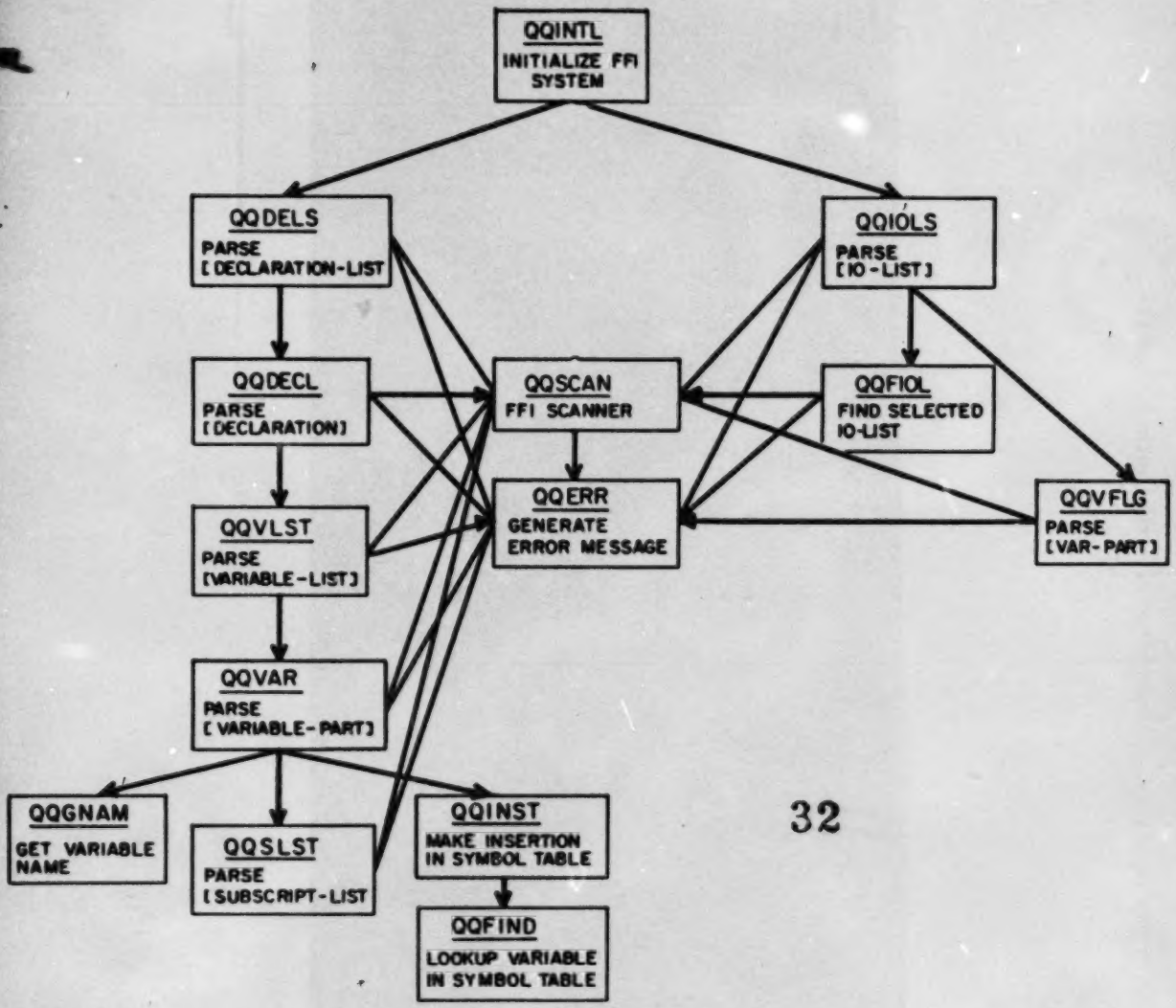
---OUTPUT FOR SUBLIST B AS WRITTEN BY QQWRIT---

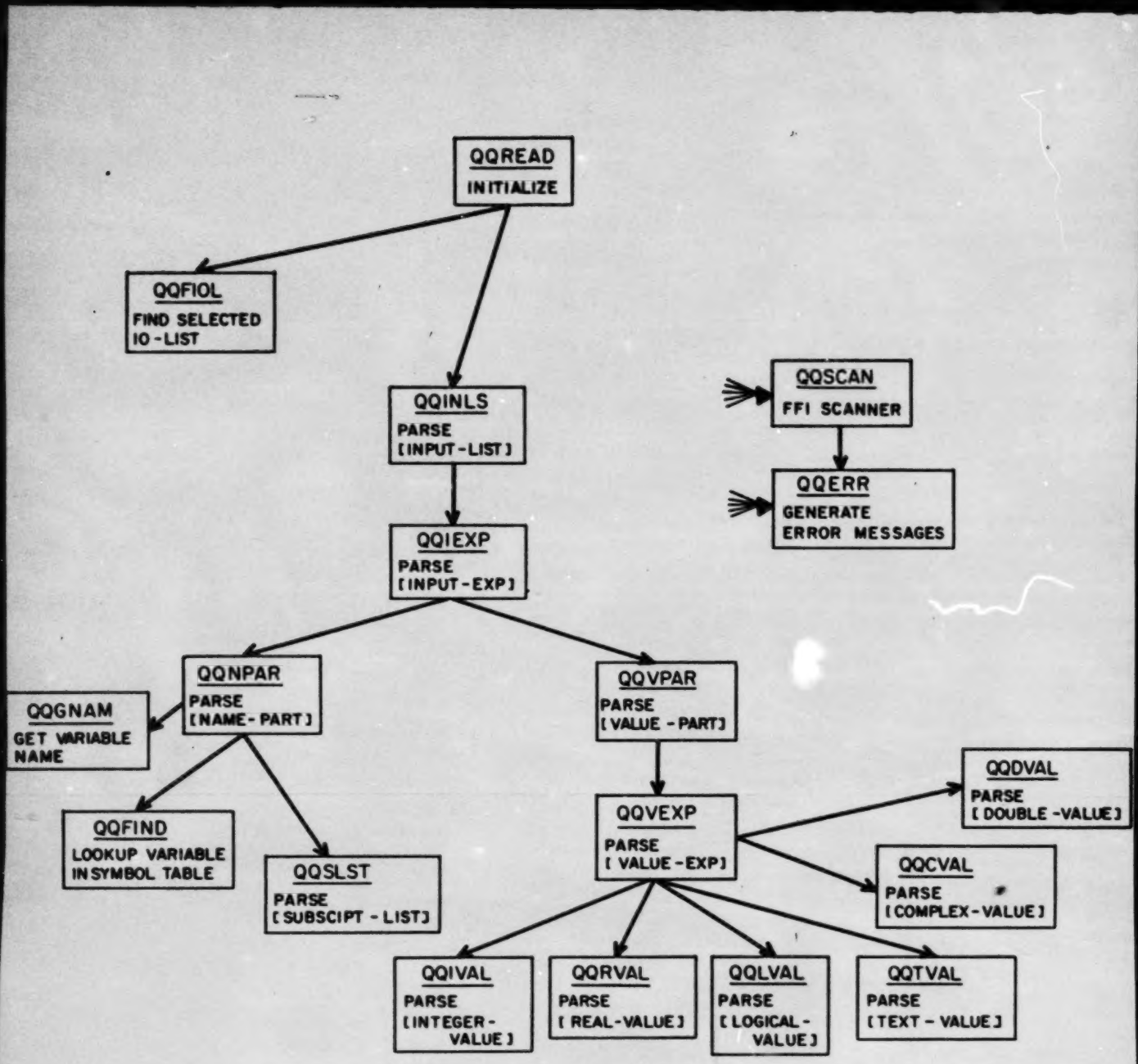
```
DX      = ( .2000000E+02,-.1800000E+02),
DY      = ( .2752000E+01, .1980000E+03),
AVG     = . .1800000D+12,  .1800000D+12,  .1800000D+12,  .1800000D+12,
          .1800000D+12,
$
```

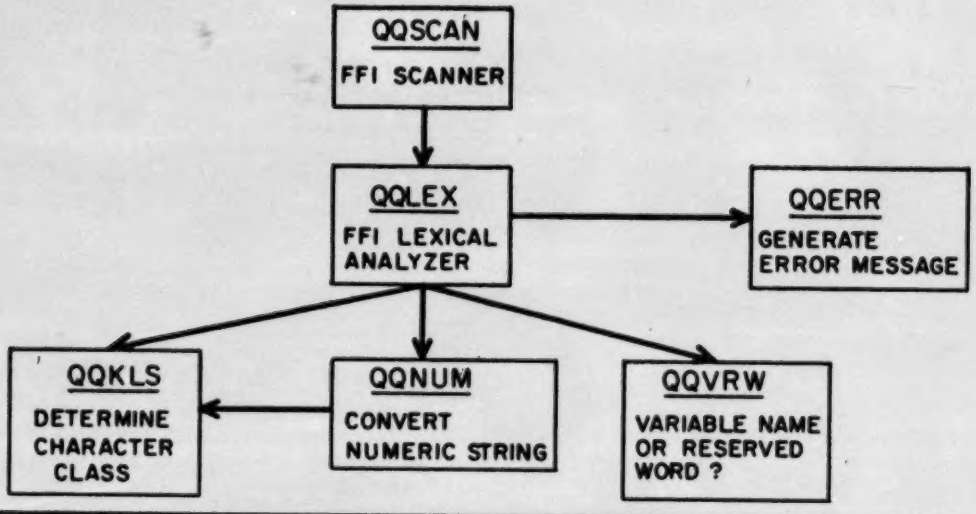
References

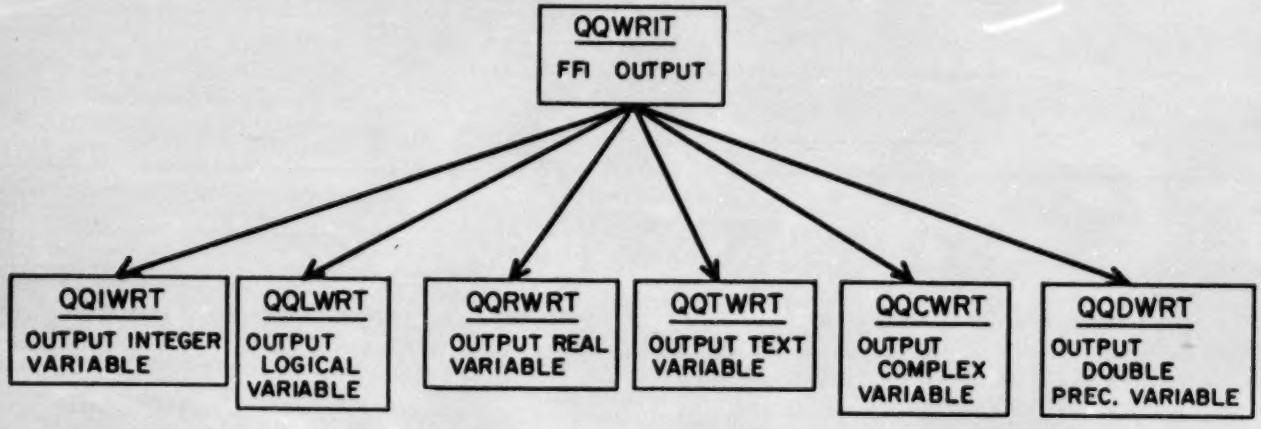
Anderson, L., and L. E. Gales. 1978. User's guide for FFORM: a format-free input system. Center for Quantitative Science in Forestry, Fisheries and Wildlife, University of Washington, Seattle.

APPENDIX I: PARSE TREE




QQREAD
INITIALIZE
QQFIOL
FIND SELECTED IO-LIST
QQINLS
PARSE [INPUT-LIST]
QQSCAN
FFI SCANNER
QQERR
GENERATE ERROR MESSAGES
QQIEXP
PARSE [INPUT-EXP]
QQNPAR
PARSE [NAME-PART]
QQGNAM
GET VARIABLE NAME
QQVPAR
PARSE [VALUE-PART]
QQFIND
LOOKUP VARIABLE IN SYMBOL TABLE
QQSLST
PARSE [SUBSCIPT-LIST]
QQVEXP
PARSE [VALUE-EXP]
QQDVAL
PARSE [DOUBLE-VALUE]
QQCVAL
PARSE [COMPLEX-VALUE]
QQIVAL
PARSE [INTEGER-VALUE]
QQRVAL
PARSE [REAL-VALUE]
QQLVAL
PARSE [LOGICAL-VALUE]
QQTVAL
PARSE [TEXT-VALUE]


QQSCAN
FFI SCANNER
QQLEX
FFI LEXICAL ANALYZER
QQERR
GENERATE ERROR MESSAGE
QQKLS
DETERMINE CHARACTER CLASS
QQNUM
CONVERT NUMERIC STRING
QQVRW
VARIABLE NAME OR RESERVED WORD ?


QQWRIT
FFI OUTPUT
QQIWRT
OUTPUT INTEGER VARIABLE
QQLWRT
OUTPUT LOGICAL VARIABLE
QQRWRT
OUTPUT REAL VARIABLE
QQTWRT
OUTPUT TEXT VARIABLE
QQCWRT
OUTPUT COMPLEX VARIABLE
QQDWRT
OUTPUT DOUBLE PREC. VARIABLE

APPENDIX II

FFORM GRAMMAR

Grammar for declarations

[declaration-list]: = [declaration] [end-of-file]
| [declaration] [end-of-line] [end-of-file]
| [declaration] [end-of-line] [declaration-list]

[declaration]: = [type] [variable-list]

[type]: = INTEGER | REAL | LOGICAL | TEXT
COMPLEX | DOUBLE PRECISION | DOUBLE

[variable-list]: = [variable-part], [variable-list]
| [variable-part], [end-of-line] [variable-list]
| [variable-part]

[variable-part]: = [variable-name]
| [variable-name] ([subscript-list])

[variable-name]: = any FORTRAN variable identifier of length less than or equal to six characters.

[subscript-list]: = [digit-list]
| [digit-list], [digit-list]
| [digit-list], [digit-list], [digit-list]

[digit-list]: = [digit] [digit] [digit-list]

[digit]: = 0|1|2|3|4|5|6|7|8|9

[end-of-file]: = $

Grammar for [io-lists]

[io-list]: = [end-of-file]
| ([io-name]) [var-part] [end-of-line] [io-list]
| ([io-name]) [var-part] [end-of-line] [end-of-file]
| ([io-name]) [var-part] [end-of-file]

[io-name]: = any single character alphanumeric identifier (nonblank)

[var-part]: = [variable-name], [var-part]

| [variable-name]

[variable-name]: = any FORTRAN variable identifier of length less than or equal to six characters

Grammar for input cards

[input]: = [ionamed-lists] | [input-list]

[ionamed-lists]: = ([io-name]) [input-list]

| ([io-name]) [input-list] [col-list] [ionamed-lists]

| [col-list] ([io-name]) [input-list]

[input-list]: = [input-exp] [end-of-file]

| [input-exp] [input-list]

| [col-list] [input-list]

[col-list]: = [end-of-line] | [end-of-line] [col-list]

[io-name]: = any single character alphanumeric identifier (nonblank)

[end-of-file]: = $

[input-exp]: = [name-part] = [value-part]

[name-part]: = [variable-name]

| [variable-name] ([subscript-list])

[value-part]: = [value-exp], | [value-exp], [col-list]

| [value-exp] [col-list]

| [value-exp], [value-part]

| [value-exp], [col-list] [value-part]

| [value-exp] [col-list] [value-part]

[variable-name]: = any FORTRAN variable identifier of length less than or equal to six characters

[subscript-list]: = [digit-list]

| [digit-list], [digit-list]

| [digit-list], [digit-list], [digit-list]

[value-exp]: = [value] | [repeat-value]*[value]

[digit-list]: = [digit] | [digit] [digit-list]

[value]: = [integer-value] | [real-value]
| [logical-value] | [text-value] | [complex-value]
| [double-precision-value]

[repeat-value]: = [digit-list]

[digit]: = 0|1|2|3|4|5|6|7|8|9

[integer-value]: = [digit-list] | + [digit-list]
| - [digit-list]

[real-value]: = [integer-value] | [integer-value].
| [integer-value]. [digit-list]
| [integer-value] [exponent]
| [integer-value]. [digit-list] [exponent]
| [integer-value]. [exponent]

[logical-value]: = T|F|.T.|.F.|.TRUE.|.FALSE.

[text-value]: = #any alphanumeric character string#

[complex-value]: = ([real-part], [imaginary-part])

[double-precision-value]: = [real-value]
| [integer-value] [d-exponent]
| [integer-value]. [d-exponent]
| [integer-value]. [digit-list] [d-exponent]

[exponent]: = E [integer-value]

[real-part]: = [real-value]

[imaginary-part]: = [real-value]

[d-exponent]: = D [integer-value]

APPENDIX III

ERROR MESSAGES

ERROR #	
1	ILLEGAL CHARACTER
2	UNRECOGNIZABLE SYNTACTIC SYMBOL
3	END-OF-CARD EXPECTED
4	RESERVED WORD EXPECTED
5	VARIABLE NAME EXPECTED
6	SYMBOL TABLE OVERFLOW
7	DUPLICATE DECLARATION
8	INTEGER SUBSCRIPT EXPECTED
9	LPAREN EXPECTED
10	RPAREN EXPECTED
11	IONAME ARGUMENT TO SUBROUTINE QQINTL DOES NOT MATCH ANY IO-LIST NAME
12	UNDECLARED VARIABLE NAME
13	EQUAL SIGN EXPECTED
14	VARIABLE DOES NOT APPEAR ON CURRENT IO-LIST
15	ONE OF SUBSCRIPTS EXCEEDS DECLARED SUBSCRIPT
16	COMMA EXPECTED
17	REPEAT VALUE MUST BE AN INTEGER
18	INTEGER VALUE EXPECTED
19	ARRAY OUT OF BOUNDS
20	REAL VALUE EXPECTED
21	TEXT VALUE EXPECTED
22	LOGICAL VALUE EXPECTED
23	DOUBLE PRECISION VALUE EXPECTED
24	COMPLEX VALUE EXPECTED
25	END-OF-FILE MARK EXPECTED